International Journal of Computer Applications (0975 – 8887)
Volume 120 – No.22, June 2015

Cost Efficient Query Optimization in Mobile Environment

Reena Kasana
Ph.D Scholar
School of Computer and Systems Sciences
Jawaharlal Nehru University, New Delhi

Saurabh Sharma
Software Engineer, Prompt Cloud
2nd Floor 118/1
80ft Road, Indira Nagar, Bangalore

ABSTRACT

Today, we live in the world of internet. With the advancement of technology, the amount of data access has increased too many folds. Internet access now is not only limited to computer devices but can now be easily accessed through mobile devices viz. Smartphones, tablets, PDA's. The internet is now available to every common man, and with its use he fires many queries on servers and uploads or downloads data from the internet. In fact, 90% of the world's data came in existence in the last three-four years, and that too because the internet is readily available to each and every common individual. Of these, much data is being uploaded and queried upon by mobile devices. As the number of devices for Internet access has increased, and so is the number of queries fired by the users on a particular server. The time taken by a query to process totally depends on the complexity involved in joining the tables distributed along the network and finally extracting the desired result out of it. Processing and optimization of various queries in mobile devices involve much join computation among data present at different sites that may be static or mobile which in turn requires much energy consumption. A mobile device has limited energy, so, it must be utilized efficiently. Much research work have been done till now, in the field of mobile computation and making efficient use of energy. However, as the mobile devices possess some asymmetric features, and because of that the old techniques used in distributed databases cannot be applied directly. This paper brings out some methods, to efficiently utilize mobile energy by employing per split semi-join using MapReduce Framework of Hadoop.

General Terms

Per-Split Semi-Join, MapReduce, Hadoop, Cost Optimization, Distributed Databases

Keywords

Per-Split Semi-Join, MapReduce, Hadoop, Cost Optimization, Distributed Databases

1. INTRODUCTION

The amount of data has grown tremendously in last two years. In fact, 90% of the world's data came into existence in the last 3-4 years. Moreover, to perform computation on such massive datasets, a very high computational time is required. Today we live in the world of internet. Most of the peoples access the internet through their mobile devices viz. Smartphones, tablets, PDA, etc. Moreover, an enormous number of queries are being fired every second by the increasing mobile device users. Data is being added every time on the internet and processing large data we require an extensive query processing. Moreover, doing computation on large data would require much energy consumption [1] [2] [3][4]. Moreover, if the querying device is a mobile device, then the energy to process these queries, which involves a high computation time is utilized from the mobile device's battery. Internet search engine's like Google, Yahoo and many more are relying on Hadoop [3] [4] [5] [6] and Map-Reduce Framework [7] to increase their efficiency and reduce search time. The main purpose of Map-Reduce is to perform computation on these large set of data. Map-Reduce systems are known for scalability, fault tolerance and flexibility to handle semi-structured and unstructured data.

Intuitively, the cost incurred in the transmission of data and energy consumed in the whole process will be reduced by employing per-split semi-join in the field of mobile computing environment. A per-split semi-join operation, which is being initiated by a mobile device, is termed a Mobile-Initiated (MI) per-split semi-join. And query initiated by the server is Server Initiated (SI) per-split semi-join. [8]

In this paper, we bring out some methods, to efficiently utilize mobile energy by employing per split semi-join using MapReduce Framework of Hadoop. Per-split semi-join is an extension of semi-join operation that takes much less time and is much efficient with the large amount of data. Also, we have discussed and compared the transfer cost and energy consumption in employing per-split semi-join operation with other operations and hence, reducing the amount of load on the mobile devices.

The rest of the paper is organized as follows. Section 2 discusses some related terminology. Our proposed work is discussed in Section 3. The obtained results after performing experiments are discussed in Section 4. Finally, we give some concluding remarks and its future aspects.

2. RELATED TERMINOLOGY AND LITERATURE SURVEY

2.1. Mobile computing environment

The recent developments in the field of mobile computing and handheld devices have made these devices capable of hosting a server even with subtle memory size [9]. If we look at the current development rate in the field of mobile technology, a mobile device is seen to have a large number of capabilities such as storage of small databases, rapid data processing capacity etc. [10]. Consequently, there would be a large number of users querying on a particular server or uploading their data on the web. Consider an example; a salesperson keeps on uploading and downloading data from the company's server. Similarly, another salesperson would also be uploading and downloading data from the server. Sometimes, there may be coherency depending upon the mechanism being employed, and the data copied to the server could be anachronistic [11]. As the most recent data is present in the mobile device, and the query generated by the salesperson may perform a series of join operation among the relations present at different sites, it might result in a very different execution scenario from the one for query processing in a traditional distributed system.

Some research work done in the field of distributed query processing is depicted in [12] [13] [14] [15]. However, these work were not able to depict the clear picture of mobile environment interaction and thereby the asymmetric features. Authors in [16] [17] present the model of pipelined and parallel processing large scale mobile data using Hadoop. These asymmetric features are explained below, and the proposed algorithm has been designed keeping these asymmetric features in mind. Moreover, the most important criteria i.e. energy conservation was not been dealt with in the traditional distributed databases. Resulting, these algorithms corresponding to distributed query processing schemes were not applicable to the mobile computing environment. The three basic asymmetric feature of mobile, which we have considered while designing the algorithm are:

- *Computing Capability between a Mobile Computer and the Server.* The servers must be continuously active and to remain active they require continuous power supply. However, continuous power supply at all the places and all the time in a mobile device is not possible. In traditional distributed databases, the nodes were considered to be all of same level, but some of them may be using mobile devices, and others may be using an uninterrupted power supply.
- *Energy Consumption While Sending and Receiving Messages*: It has been observed that the energy required in sending data packet is more than the energy needed for receiving the data set [18] [19]. This feature is to be considered while designing some algorithms related to the mobile environment.
- *Active Mode Energy Consumption by Mobile Computers:* Mobile computer's idleness state is also to be considered. Active mode power consumption of a mobile device is always greater than the device in ideal mode [19] [20]. So, a mobile computer must be designed in such a way that the workload to be done on Mobile Device is transferred to the server, and the Mobile device can always remain in Ideal mode.

2.2. Hadoop

Various big companies like Yahoo, Google, and Facebook, etc. have to deal with large amount of data. Moreover, with each query being generated by the user, they are required to process a large amount of data in a very less time. These companies have their data distributed at different sites located at various places in the world. Google's File System (GFS) [21] and Big Table [22] are examples of such distributed file systems. These systems are a cluster of thousands of commodity machines, and these machines assure reliability, scalability and availability issues [21] [22] [23]. To reduce input/output, each file in these storage systems is divided into chunks or blocks of data and each block is present at the different site. When some query is fired, parallel processing is done on all these blocks.

Hadoop is open source software developed by Doug Cutting, who also developed Apache Lucene, the most commonly used text search library. Hadoop is meant for distributed processing of large data sets distributed among different clusters of commodity servers. The primary objective of Hadoop is to join various single-node systems, forming a network using and signaling the task coordinator to synchronize the task computation at all the nodes and thereby enhancing parallel computation. Hadoop has two main subprojects: MapReduce and Hadoop Distributed File System (HDFS) [24]. In our proposed work, we have focussed on Map-Reduce only.

2.3. Map-Reduce

Several algorithms have been proposed till now for evaluation of operations in the field of parallel and distributed Relational DBMS [25] [26] [27]. However, with the rapid increase in data in last few years, Map Reduce based systems are the best-suited alternative to getting acceptable performance [28]. Dean et. al in [29] introduced Map Reduce in the year 2004. Map Reduce does parallel processing on the data which is distributed among various nodes. Hassan et. Al in [30] [31] tried to explain the working of Map Reduce by using certain algorithms. The working of Map-Reduce is divided into three main operations. First is the Map operation, where parallel processing is done on each node locally. Next, the data is replicated on various nodes in the cluster. Moreover, finally, the output of Map operation is fed as the input to the Reduce operation. Google's Map Reduce model is based on two functions and is presented in [29] [21]. The two function viz. Map and Reduce signatures are as given below:

$$\boldsymbol{Map}: \quad (Key_1, n_1) \rightarrow list\ (Key_2, n_2)$$

$$\boldsymbol{Reduce}: \quad (Key_2, list\ (n_2)) \rightarrow list\ (n_3)$$

The user-written Map function takes two values Key: Key_1 and the corresponding frequency value n_1 and outputs a list of intermediate Key/Value pairs $(\mathrm{Key}_2, \mathrm{n}_2)$. The intermediate Key/Value pairs are partitioned according to Key Key_2, where all the pairs have the same value for Key_2, and belong to the same group.

Map Reduce is best known for its fault tolerance, reliability, scalability and ability properties to operate in the heterogeneous environment. Map Reduce model is mainly preferred for homogeneous datasets. It is not much feasible to perform join operation where heterogeneous datasets need to be merged [32]. However, applying the homogenization process, it can still be used for heterogeneous datasets [28]. Fault tolerance is achieved by recognizing the failed jobs and reassigning it to other resources.

Some of the previous studies [33] lead to some performance related issues in Map Reduce. Most of the performance enhancing tools or functions used in database systems like view, stored procedure, etc. cannot be used because it does not allow to the data to be modelled, thereby it could not be loaded before pre-processing of the data.

2.4. Per-split semi join

Semi join has also been implemented using map reduce framework in [34]. One of the main problems with Semi Join is that not every record in the filtered version of map reduce framework gets joined with records of the other table. Per Split Semi-Join is an extension of Semi-Join. Per Split Semi Join is divided into three phases. The first phase and the third phase is the map only phase while the second phase is the map-reduce phase. In the first phase, i.e. map only phase, the input table is divided into many splits. Each split holds a certain number of records. Each split is read by a mapper, and each mapper generates a file Fi.uk, which holds a list of unique keys from the split read by the mapper. On the basis of these keys, join operation is performed. In the second phase, records from the other table are brought into main memory by the mapper module. This phase has two functions viz. Init () and Close (). The Init () function loads the keys of the file Fi.uk into the hash table. The close () function matches each unique key and for each matching record found it projects the result into the hash table. Matched records are tagged with the table they belong to, which is used by the reduce phase to summarize all the output. In the final phase, the files generated for both the tables are joined, which gives the result of per split semi join. The result generated is sent back to the initial node to perform the join operation.

3. PROPOSED WORK

Here, we proposed a method for reducing the cost of join methods and hence saving energy in mobile computing. Overview of join processing in mobile computing is depicted in figure 1. Here server has the relation S, and the mobile unit holds the relation M. Let the mobile user put a query on a server that requires join operation between S and M relations.

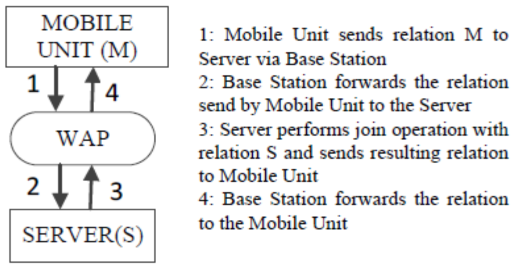
MOBILE UNIT (M)
1
4
WAP
2
3
SERVER(S)
1: Mobile Unit sends relation M to Server via Base Station
2: Base Station forwards the relation send by Mobile Unit to the Server
3: Server performs join operation with relation S and sends resulting relation to Mobile Unit
4: Base Station forwards the relation to the Mobile Unit

Figure 1. Join Processing in Mobile

The mobile user sends the relation M to the server S where join operation is performed, and the result is sent back to the mobile unit. The mobile user sends the relation M to the server through the WAP gateway, where the server address is recognized, and the various algorithms are applied to find the shortest path to the server. When the relation M reaches the server, the corresponding tuples are matched having the same join key as the relation M. The desired result is again sent back to the WAP gateway, where once again the same procedure is repeated and finally the result is sent to the mobile user.

3.1. The cost model

Cost model for the proposed algorithm performing the join operation through the MapReduce framework is derived as in [35]. Let consider two relations A and B. and we need to compute $A \bowtie B$ with the help of per split. As the property goes:

$$A \bowtie B = A \bowtie (B \ltimes A) = (A \ltimes B) \bowtie (B \ltimes A)$$

The two relations A and B are taken as input, which are divided into various splits of the file which are stored in distributed manner at different nodes of Hadoop. Some of the notations used throughout the paper are:

• $S_{A,i}$	: Split i of relation A
• $\lvert S_{A,i}\rvert$	: Number of records in split i of relation A
• t_L	: Time taken to load data of a node/split
• t_R	: Time taken to read data by the mapper
• t_W	: Time taken to write a data into B^+ tree
• t_C	: Time taken to compare data with the existing Key in B^+ tree
• t_I	: Time taken to create an index on a data.
• $N(A)_m$	: No of mappers involved in relation A
• $N(A)_r$	: No of reducers mappers involved in relation A
• M_P	: Message Protocol cost
• M_L	: Message latency cost
• ρ	: Selectivity Factor
• $LK(B)_{i,k}$	: The number of local keys of relation B generated by i mapper and being fed to k reducer
• GBT	: Global B^+ Tree holding the index of final join keys.

As discussed in section 2.4, Per Split Semi Join is divided into three phases. We will discuss each phase one by one.

3.1.1. Phase1

There are two groups of mappers; one manages the chunks of relation A and other manages the chunks of relation B. These chunks are the splits of relation A and B stored in DFS at different nodes in Hadoop File System. Each group of mapper reads the split(s) assigned to it. All the unique key elements with their frequency count are stored with hash indexes in the form of B+ Trees. At the end of this phase, each mapper outputs a file, Fi.uk containing a set of unique keys with their frequency count. The time/cost calculation of this phase is given in equation 1:

$$O\left(max\left(\sum_{i=0}^{N_m}\left(S_{A,i} * t_L + \left|S_{A,i}\right| * (t_R + t_C + t_W + t_I)\right)\right), \left(\sum_{i=0}^{N_m}\left(S_{B,i} * t_L + \left|S_{B,i}\right| * (t_R + t_C + t_W + t_I)\right)\right)\right) \text{ ------ (1)}$$

Where $S_{A,i} * t_L$ denotes the cost/time of loading a single split *i in* the memory hash table. $\left|S_{A,i}\right| * (t_R + t_C + t_W + t_I)$ Indicates the time taken to read the join keys, compare with existing keys, writing the data and creating indexes for the relation A. and $S_{A,i}$ indicates the number of tuples in a particular split *i* of relation A.

3.1.2. Phase2

Each mapper of the relation A outputs a file Fi.uk which contains a set of unique keys on which join will take place. The unique keys identified by the mappers of relation B stored in form of B+ tree along with the unique key file Fi.uk brought as output by the mappers of relation A, are fed as input to the reducer. Each reducer reads the corresponding split of relation B and all the files generated by relation A mappers. On applying the same hash function as applied on all the mappers in phase 1 while partitioning (key, n), allows us to forward all entries having same value of key to partitions of all local B+ trees holding the same index on all the mappers. So, each reducer fetches the associated records from the mappers of relation B along with the files generated by the mappers of A relation. Each reducer updated the frequency count of the keys.

The cost of forwarding the files Fi.uk and the corresponding records of relation B to the reducers is given in equation 2.

$$O\left(\left(N(A)_m * (M_P + M_L)\right) + \left(\sum_{k=0}^{max\,z}\sum_{i=0}^{N_m}\{LK(B)_{i,k} * (M_P + M_L)\}\right)\right) \text{ ------ (2)}$$

Where $N(A)_m$ represent the number of mappers in relation A. As each mapper produces a file Fi.uk at the end of phase 1 and these files are sent to the reducer. So the number of mappers is equal to the number of files generated. $(M_P + M_L)$ represents the message protocol cost and message latency. $\{LK(B)_{i,k} * (M_P + M_L)\}$ Gives the cost of forwarding keys of relation B generated by ith mapper and fed to k reducer.

To prove the scalability of our approach, for the first half i.e. sending the relation to the corresponding set of mappers, we consider the equation 3 as the lower bound to our cost model.

$$\mathrm{O}\left(\left(N(A)_m * \ (M_P + \ M_L)\right) + min\left(|GBT| * \ (M_P + \right.\right.$$
$$ML, BNm{*}MP{+}\ ML \quad \text{------ (3)}$$

The cost of receiving the records from the corresponding mapper of relation B and the files Fi.uk from the mappers of relation A is given in equation 4:

$$\mathrm{O}\left(\left(N(A)_m * \ (M_L)\right) + \ \left(\sum_{k=0}^{max\,z} \sum_{i=0}^{N_m} \{LK(B)_{i,k} * \ (M_L)\}\right)\right) \quad \text{------ (4)}$$

Here, we propose scalability of our approach for the phase in which the records from the respective mappers are fed into corresponding reducers. The equation 5 proposes the lower bound cost model for this phase:

$$\mathrm{O}\left(\left(N(A)_m * \ (M_P + \ M_L)\right) + min\left(\frac{N_m}{N_r} * \left(|GBT| * \ M_P + \right.\right.\right.$$
$$GBT{*}\ tR{+}\ tI, NmNr{*}\ B{*}\ MP{+}\ B{*}\ tR{+}\ tI \quad \text{------ (5)}$$

So, the total time/cost of this phase is given by equation 6.

$$\mathrm{O}\left(\left(\left(N(A)_m * \ (M_P + \ M_L)\right) + min\left(\frac{N_m}{N_r} * \left(|GBT| * \ M_P + \ ||GBT|| * \ (t_R + \ t_I)\right), \frac{N_m}{N_r} * \ \left(|B| * \ M_P + \ ||B|| * (t_R + \ t_I)\right)\right)\right) + \ \left(\left(N(A)_m * \ (M_P + \ M_L)\right) + min\left(\frac{N_m}{N_r} * \left(|GBT| * \ M_P + \ ||GBT|| * \ (t_R + \ t_I)\right), \frac{N_m}{N_r} * \ \left(|B| * \ M_P + \ ||B|| * \ (t_R + \ t_I)\right)\right)\right)\right) \quad \text{------ (6)}$$

3.1.3. Phase3

This is the last phase called join phase. In this phase, the index of B+ Tree extracted is sent to the site where the result of join is to be sent. So a direct join is performed. The final cost of this phase is given by equation 7.

$$\mathrm{O}\left(|GBT| * \ \sum_{i=0}^{N(A)_m} |S_{A,i}| * (M_P + \ M_L)\right) \quad \text{------ (7)}$$

Moreover, the cost of receiving these at the nodes is given by equation 8:

$$\mathrm{O}\left(|GBT| * \ \sum_{i=0}^{N(A)_m} |S_{A,i}| * (M_P)\right) \quad \text{------ (8)}$$

Table 1. **Algorithms for Per-Split Semi Join**

Algorithm 1: Per Split Semi- Join algorithm workflow

Each Mapper
- *Reads the assigned split of both the relation from the DFS*
- *Partitions the data according to the join key and also maintains the frequency count of each key element*
- *Unique key generated from each split is stored in file F_i.uk*

Each Reducer
- *Remotely reads the local B^+ tree partitions holding its index for the relation B and all the F_i.uk files generated after map phase of relation A.*
- *Each reducer checks for the matching key from the file F_i.uk. If no match found, an entry is added else the frequency is updated.*
- *Each reducer generated a global B^+ tree.*
- *Merging of all the global trees is done.*

Join
- *The indexes of the keys are used to fetch the desired tuples, and direct join is performed.*

Table 2. **Algorithm for Map and Reduce operations**

Algorithim2 Map Function for computing local B^+ Tree	**Algorithm 3** Reduce function
Map(char relation, const char* key){* */* relation : is the relation name, key : the join attribute*/ Create a B^+ tree.* ***For** tuple t in split i of relation {* *Hash t into the memory hash table by applying hash_function(t.v);* ***if** join key is already present in the B^+ tree **then** increment the frequency of the particular key in B^+ tree **else*** *add the entry (key,1) to the B^+ tree*	*Reduce (int reducer_id, DS B+ Tree, File Fi.uk) {* */*reducer_id : id of the reducer, Fi.uk : files generated in phase 1*/* *Create global B+ tree* ***For** each mapper in relation B {* *Read the local B+ tree corresponding to each reducer_id and the keys in files Fi.uk* *For each pair in B+ tree {*

Endif
  }
}
	If key already exists in global B+ tree ***then****,*
	    Update frequency of the corresponding key as freqv += nv
	  Endif
	  }
	}

4. RESULTS AND DISCUSSION

4.1. Notations and assumptions

The experiment conducted involves some notation and assumption. The cardinality of a relation R is denoted by $|R_i|$ and an attribute's cardinality is being denoted by $|A|$. W_A Denote the size of the attribute A. The notation $R_i \ltimes R_j$ and $R_i \diamond R_j$ denotes the Semi Join and Per Split Semi Join operation respectively between relations R_i and R_j. And let $|R_i \ltimes R_j|$ and $|R_i \diamond R_j|$ denote the cardinality of Semi Join and Per Split Semi Join respectively. Let ρ denote the selectivity factor of a particular attribute in a relation R_i.

Let W_A denote an attributes width and correspondingly W_R denotes the width of a relation's tuple. So the total size of the file can be calculated as $W_R * |R_i|$. For any join operation we define selectivity factor $\rho_{i,a}$ for an attribute A as $\frac{|R_i \ltimes R_j|}{|R_i|}$ (for Semi-Join operation) and $\frac{|R_i \diamond R_j|}{R_i}$ (for Per Split Semi-Join operation).

4.2. Cost model for join operation

Now we will derive the cost model, keeping asymmetric features of mobile discussed in section 2.1, in mind. The cost model being derived will lead to a better Join methodology and hence, enhance query processing by reducing the amount of data transferred between different sites. To project the cost and energy involved in data transmission from one node to another, some assumptions were made. Let e_{RC} and e_{SD} denote the amount of energy consumed in receiving a tuple from a relation R. r_E denote the send-receive energy ratio. The value of r_E will always be greater than 1, because the energy involved in receiving a relation is always greater than the energy involved in sending the relation. Moreover, the send-receive energy ratio can be assumed to lie in between the range of 2-10 [36]. If the cardinality of a relation is $|R|$. Then the total energy involved in sending the relation R is $e_{SD} * |R|$. Let $T_s(R_i \ltimes R_j)$ and $T_s(R_i \diamond R_j)$ denote the total time spend in processing Semi Join and Per Split Semi Join respectively. Let t_{tuple} denote the processing time per tuple in a relation.

If r_{SM} denote the mobile-server processing ratio then correspondingly the time spend in processing Semi Join and Per Split Semi Join can be denoted as $\frac{1}{r_{SM}} * T_s(R_i \ltimes R_j)$ and $\frac{1}{r_{SM}} * T_s(R_i \diamond R_j)$. As the processing capacity of mobile is less than that of home computer or server, hence its value will always remain less than 1.

Now, we calculate the estimated cost of the transfer of data and the overall energy consumed in the scenario. Let R_i and R_j be two relations present at Mobile and Server respectively. Let $D_{t,PSSj}$ and $D_{t,Sj}$ denote the amount of data transfer on employing Per Split Semi Join and Semi Join respectively. Similarly, $E_{t,PSSj}$ and $E_{t,Sj}$ denote the amount of energy consumed at Mobile end in processing Per Split Semi Join and Semi Join respectively.

4.2.1. Cost computation when join processing is done at Mobile End

In this case, the server sends the particular join key attribute of relation R_j to the Mobile Unit. Suppose the Join operation is done on the basis of attribute 'A' of relations R_i andR_j. Then

- For Semi Join:

$$D_{t,SJ} = |A| * w_A + \rho_{Rj,A} * |R_j| + |R_i \ltimes R_j|$$

$$E_{t,Sj} = e_{RC} * |A| + \frac{1}{r_{SM}} * T_S(R_i \ltimes R_j) + e_{SD} * \rho_{Rj,A} * |R_j| + e_{RC} * |R_i \ltimes R_j|$$

- For Per Split Semi Join:

$$D_{t,SJ} = \rho_{Ri,A} * |A| * w_A + \rho_{Rj,A} * |R_j| + |R_i \diamond R_j|$$

$$E_{t,SJ} = e_{RC} * \rho_{Rj,A} * |A| + \frac{1}{r_{SM}} * T_S(R_i \diamond R_j) + e_{SD} * \rho_{Rj,A} * |R_j| + e_{RC} * |R_i \diamond R_j|$$

4.2.2. Cost computation when join processing is done at Server End

In this case, the Mobile user sends the particular join key attribute of relation R_i to the Server. Suppose the Join operation is done on the basis of attribute 'A' of relations R_i and R_j . Then:

- For Semi Join

$$D_{t,SJ} = |A| * w_A + \rho_{Rj,A} * |R_j|$$

$$E_{t,SJ} = e_{SD} * |A| + r_{SM} * T_S(R_i \ltimes R_j) + e_{RC} * \rho_{Rj,A} * |R_j|$$

- For Per Split Semi Join:

$$D_{t,SJ} = \rho_{Ri,A} * |A| * w_A + \rho_{Rj,A} * |R_j|$$

$$E_{t,SJ} = e_{RC} * \rho_{Rj,A} * |A| + r_{SM} * T_S(R_i \diamond R_j) + e_{SD} * \rho_{Rj,A} * |R_j|$$

5. IMPLEMENTATION

After theoretical evaluation of both the join techniques, experiments were performed on both the techniques by analyzing for different selectivity factor with data set of various sizes. The datasets were generated by a program, keeping in mind the value of selectivity factor for which the experiment is to be performed. Files for different sizes were generated for the same selectivity factor and were placed in various nodes of a network, forming a cluster. The experiment was performed one after the other, on the same set of nodes, so as to reduce the errors with respect to different processing capabilities of different nodes. Table 3 and Table 4 specifies the time taken (in seconds) to get the results for joining two relations, distributed along the network using per split semi-join and semi-join respectively.

Table 3. Experimental results for Per-Split Semi-Join (Time in Seconds)

		Selectivity Factor									
		0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
Size (MB)	**1**	111	94	90	96	88	90	93	90	89	88
	5	98	94	94	93	91	89	92	92	92	92
	10	96	94	95	89	90	91	89	90	88	90
	15	106	89	89	90	89	88	89	90	89	89
	20	105	94	87	89	91	91	89	88	90	90
	25	93	91	89	113	88	88	90	98	92	89
	30	95	128	91	91	90	90	95	100	94	93
	35	96	99	91	107	91	90	91	89	99	89
	40	138	97	92	137	90	132	96	89	90	92
	45	98	93	95	93	94	95	93	93	91	94
	50	137	193	95	97	92	92	94	105	94	88

Table 4. Experimental results for Semi-Join (Time in Seconds)

		Selectivity Factor									
		0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
Size (MB)	**1**	74	75	75	74	74	74	75	74	76	74
	5	102	110	110	107	107	106	109	107	100	106
	10	127	129	129	133	129	134	131	133	129	127
	15	151	148	148	150	148	148	149	156	155	154
	20	174	174	174	175	179	183	176	178	179	180
	25	204	205	205	212	209	200	249	224	208	205
	30	241	236	236	240	242	240	242	242	237	270
	35	258	263	263	264	260	262	263	258	259	264
	40	308	305	305	307	310	306	306	311	311	314
	45	336	334	334	329	329	333	331	329	329	332
	50	357	354	354	350	352	354	353	354	352	354

Using the data obtained after performing experiments, graphs were plotted to understand the clear difference in processing times. The graph figure 3 shows the comparison of both the techniques. Further, the graphs Figure 2, 4-7 shows the comparison of both the techniques simultaneously, for a particular selectivity factor.

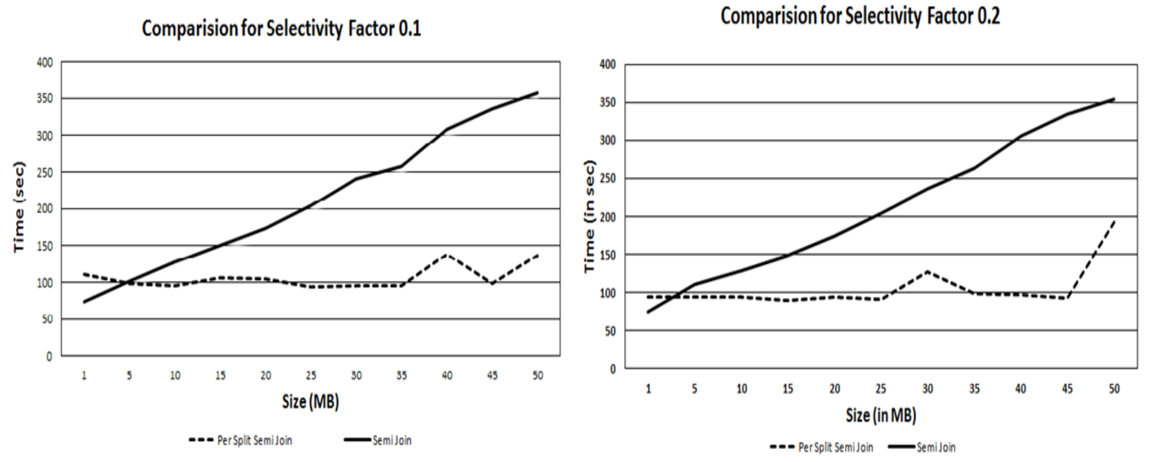

Figure 2. Comparison of Selectivity Factor 0.1 and 0.2

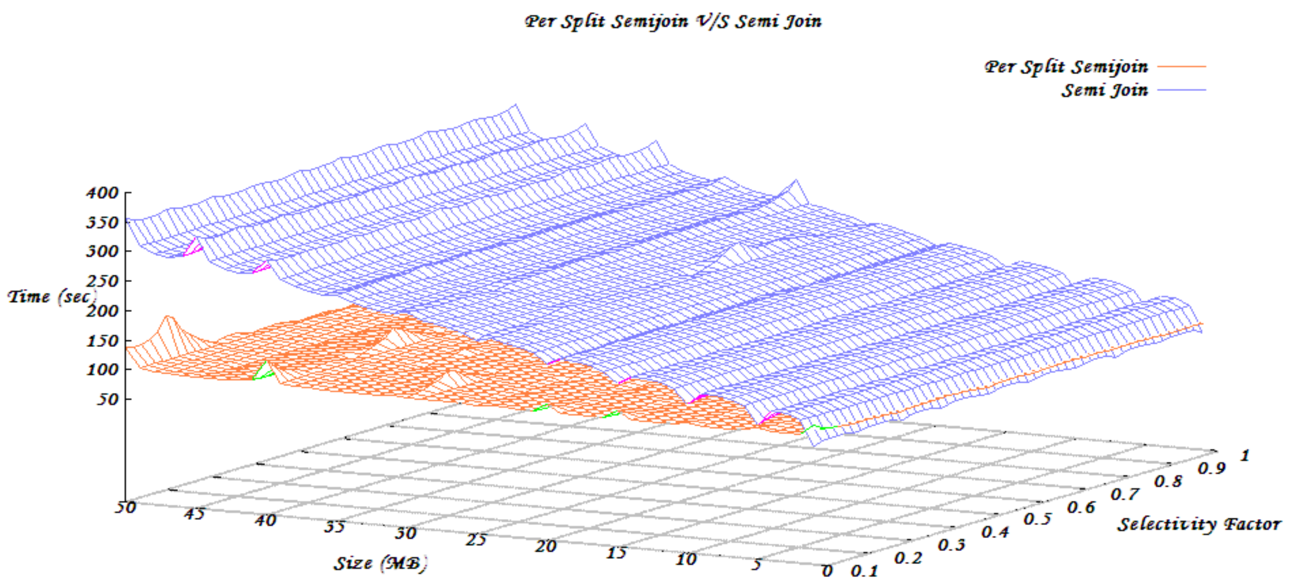

Figure 3. Comparison of Per-Split Semi Join and Semi-Join

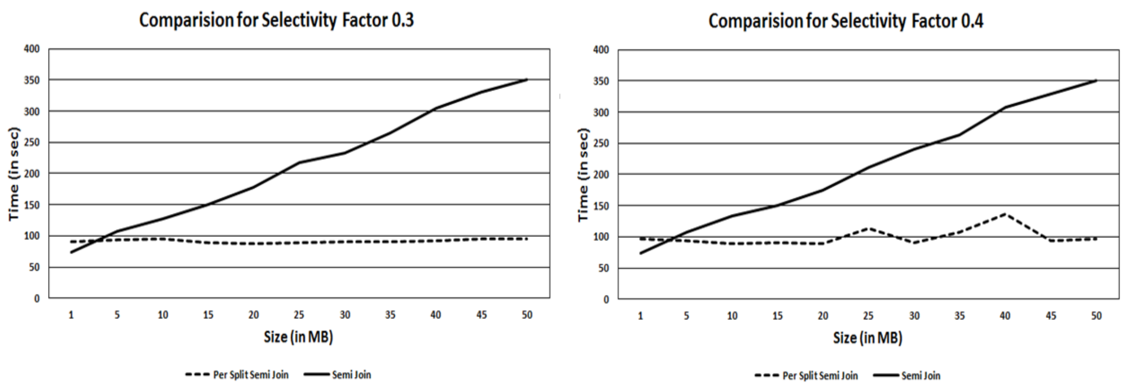

Figure 4. Comparison of Selectivity Factor 0.3 and 0.4

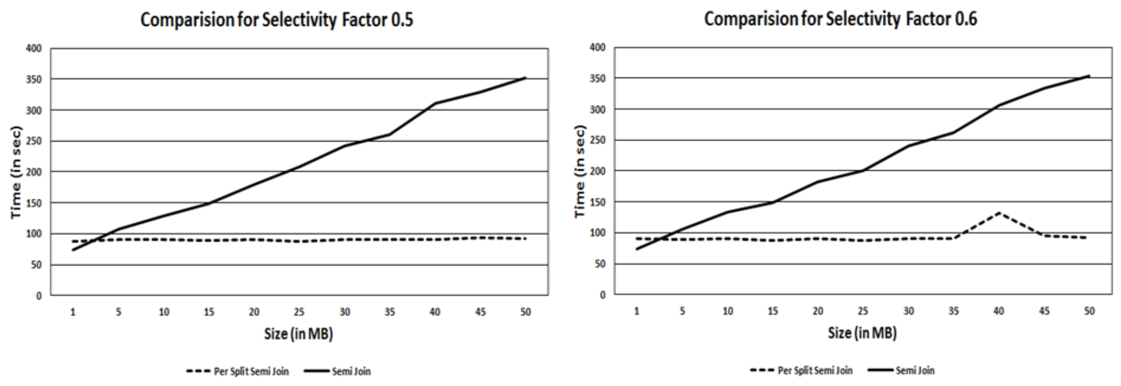

Figure 5. Comparison of Selectivity Factor 0.5 and 0.6

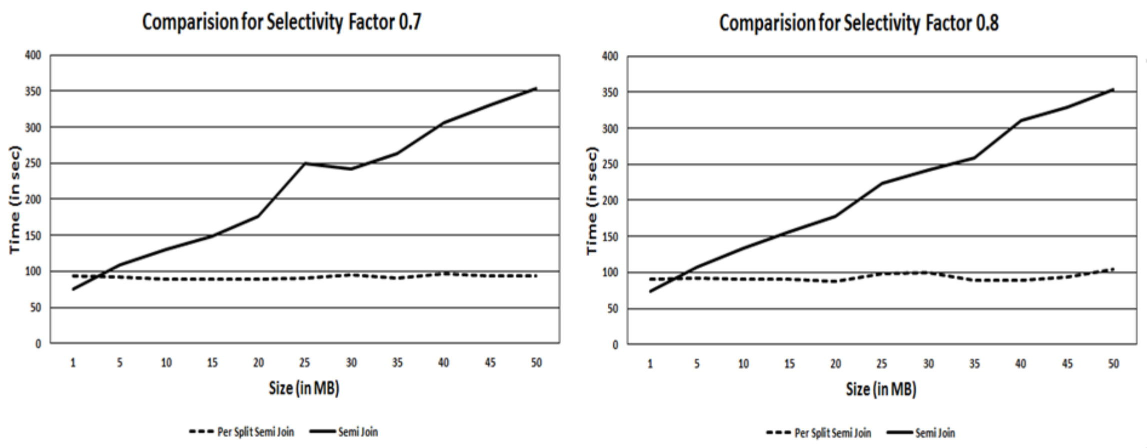

Figure 6. Comparison of Selectivity Factor 0.7 and 0.8

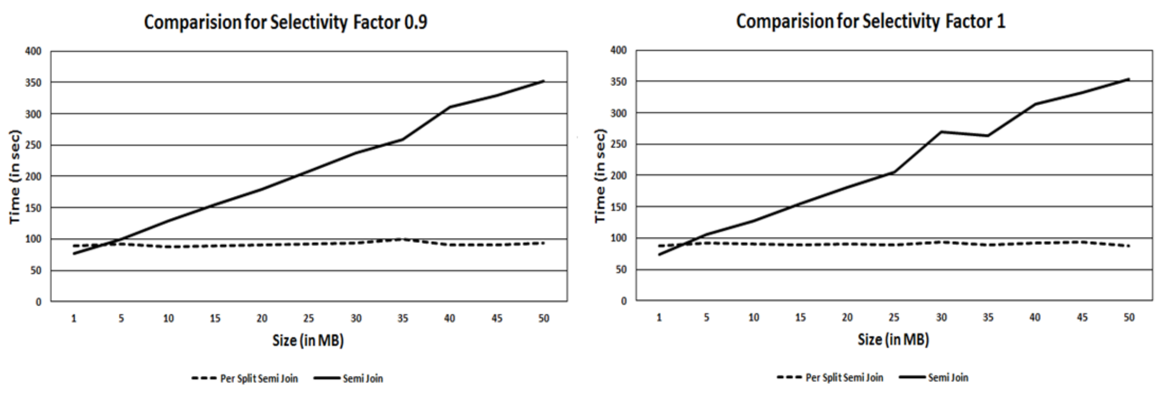

Figure 7. Comparison of Selectivity Factor 0.9 and 1.0

6. CONCLUSION AND FUTURE WORK

With the result of the experiments performed and our theoretical evaluations, we come to a conclusion that, the cost and time taken to perform a Per-Split Semi Join operation is far much better than Semi-Join operation. As the whole experiment was conducted using MapReduce operations on Hadoop clusters, which further concludes that MapReduce based systems are scalable, fault-tolerant and are easily programmable. They automatically take care of parallel computation and hence reduce the computation time. Against all the existing algorithms based on hashing like some of them are presented in [35], our algorithm is insensitive to data skew. In the presence of skewed data, and right choice of the hashing function, doing join and semi-join computations are very efficient. In this paper, we have computed join of two relations using the concept of per split semi join through Map Reduce framework of Hadoop.

As we have seen that per-split semi-join is better that semi-join operation as it consumes less power, which would be beneficial for the mobile environment, and it will save the mobile device's power. But still the use of the mobile device as a distributed database is an ongoing research topic, which requires further more extensive study in this area, to implement things like per-split semi-join on mobile devices.

7. REFERNCES

[1] D. Barbara, "Mobile Computing and Databases - A Survey," IEEE Transactions on Knowledge and Data Engineering, vol. 11, no. 1, pp. 108-117, January/February 1999.

[2] M. A. H. Hassan and M. Bamha, "Semi-join Computation on Distributed File System Using Map-Reducr-Merge Model," ACM, 22-26 March 2010.

[3] K. Shvachko, H. Kuang, S. Radia and R. Chansle, "The Hadoop Distributed File System," in Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010.

[4] J. Venner, Pro Hadoop, Apress, June 22, 2009.

[5] T. White, Hadoop: The Definitive Guide, O'Reil.

[6] K. Karun A. and C. K., "A review on hadoop — HDFS infrastructure extensions," in Information & Communication Technologies (ICT), 2013 IEEE Conference, pp. 132-137, April 11-12, 2013. .

[7] R. M. Arasanal and D. U. Rumani, "Improving MapReduce Performance through Complexity and Performance Based Data Placement in Heterogeneous Hadoop Clusters," 9th International Conference, ICDCIT 2013, vol. 7753, no. 1, pp. 115-125, February 5-8, 2013 .

[8] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita and Y. Tian, "A Comparison of Join Algorithms for Log Processing in Map-Reduce," in SIGMOD'10, Indianapolis, Indiana, USA., June 6–11, 2010.

[9] "Applications of Mobile Computing," 2005. [Online]. Available: http://www.nokia.com/3g/ index.html.

[10] "Palm Pilots of 3com," 2005. [Online]. Available: http://www.palm.com.

[11] P. S. Yu, M. S. Chen and T. H. Yang, "On Coupling Multiple Systems with a Global Buffer," IEEE Trans. Knowledge and Data Engineering, vol. 8, no. 2, pp. 339-344, April 1996.

[12] P. S. Yu and M. S. Chen, "Interleaving A Join Sequence with Semijoins in Distributed Query Processing," IEEE Trans. Parallel and Distributed Systems, vol. 3, no. 5, pp. 611-622, September 1992.

[13] P. S. Yu and M. S. Chen, "Combining Join and Semi-Join Operations for Distributed Query Processing," IEEE Trans. Knowledge and Data Engineering, vol. 5, no. 3, pp. 534-542, June 1993.

[14] M. J. Franklin, B. T. Jonsson and D. Kossman, "Performance Tradeoffs for Client-Server Query," ACM SIGMOD International Conference on Management of Data, pp. 149-160, June 1996.

[15] C. Wang and M. S. Chen, "On the Complexity of Distributed Query Optimization," IEEE Transactions on Knowledge and Data Engineering, vol. 8, no. 4, pp. 650-662, August 1996.

[16] M. Koca, I. Ari, U. Kocak, O. Calikus and C. Sezgin, "Parallel and Pipelined processing of Large Scale Mobile communication data using Hadoop open-source framework," in 20th conference on Signal Processing and Communications Applications Conference (SIU), 2012, April 18-20, 2012.

[17] O. Choi, W. Jung, K. Kim and H. Yeh, "Mobile Cloud Computing Model for Data Processing," in 6th International Conference on New Trends in Information Science and Service Science and Data Mining (ISSDM), 2012, October 23-25, 2012.

[18] A. Datta, D. E. VanderMeer, A. Celik and V. Kumar, "Broadcast protocols to support efficient retrieval from databases by mobile users," ACM Transactions on Database Systems (TODS), vol. 24, no. 1, pp. 1-79, March 1999.

[19] R. Jain and N. Krishnakumar, "An asymmetric cost model for query processing in mobile computing environments," Wireless Information Networks, vol. 351, no. 1, pp. 363-377, 1996.

[20] R. Jain and N. Krishnakumar, "Asymmetric Costs and Dynamic Query Processing in Mobile Computing," 5th WINLAB workshop, April 1995.

[21] S. Ghemawat, H. Gobioff and S. T. Leung, "The Google File System," in SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems, The Sagamore, Bolton Landing (Lake George), October 19-22, 2003.

[22] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes and R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," OSDI '06: 7th USENIX Symposium on Operating Systems Design and Implementation, Berkeley, CA, USA, vol. 5, no. 1, pp. 205-218, 2006.

[23] "Apache Hadoop," [Online]. Available: http://hadoop.apache.org/core/.

[24] J. Xu and J. Liang, "Research on Distributed File System with Hadoop," Internatiaonal Journal of Communications in Computer and Information Science, vol. 345, no. 1, pp. 148-155, December 7-9, 2012.

[25] M. Bamha, "An Optimal Skew-insensitive Join and Multi-join Algorithm for Distributed Architectures," International Conference on Database and Expert System Applications, vol. 3588, no. 1, pp. 616-625, August 22-26, 2005.

[26] M. Bamha and G. Hains, "A Skew-Insensitive Algorithm for Join and Multi-join Operations on Shared Nothing Machines," International Journal of Database and Expert Systems Applications, vol. 1873, no. 1, pp. 644-653, 2000.

[27] M. Bamha and G. Hains, "Frequency-Adaptive Join for Shared Nothing Machines," Journal of Parallel and Distributed Computing Practices (PDCP), vol. 2, no. 3, pp. 333-345, September 1999.

[28] H. C. Yang, A. Dasdan, R. L. Hsiao and D. S. Parker, "Map-reduce-merge: simplified relational data processing on large clusters," Proceedings of the 2007 ACM SIGMOD international conference on Management of data, New York, USA, pp. 1029-1040, 2007.

[29] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in Proceedings of the 6th conference on symposium on operating systems design and implementation, 2004.

[30] M. A. Hassan and M. Bamha, "Semi-join computation on distributed file systems using map-reduce-merge model," SAC '10 Proceedings of the 2010 ACM Symposium on Applied Computing, pp. 406-413, 2010.

[31] H. H. Le, S. Hikida and H. Yokota, "NameNode and DataNode Coupling for a Power-Proportional," 18th International Conference, DASFAA 2013, vol. 7826, no. 1, pp. 99-107, April 22-25, 2013.

[32] R. Pike, S. Dorward, R. Griesemer and S. Quin, "Interpreting the data: Parallel analysis with Sawzall," Scientific Programming - Dynamic Grids and Worldwide Computing, vol. 13, no. 4, pp. 277-298, October 2005.

[33] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden and M. Stoneb, "A comparision of approaches to Large-Scale Data Analysis," in SIGMOD'09, June 29-July 2, 2009.

[34] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita and Y. Tian, "A comparison of join algorithms for log processing in MaPreduce," SIGMOD '10 Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 975-986, 2010.

[35] K. Shvachko, H. Kuang, S. Radia and R. Chansl, "The Hadoop Distributed File System," in Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010.

[36] S. Ganguly and R. Alonzo, "Query Optimization in Mobile Environments," 5th Workshop on Foundations of Models and Languages for Data and Objects, pp. 1-17, September 1993.

[37] M. H. Dunham and A. Helal, "Mobile Computing and Databases: anything new?," ACM SIGMOD Record, vol. 23, no. 4, pp. 5-9, December 1995.

[38] J. Jing, A. S. Helal and A. Elmagarmid, "Client-Server computing in mobile environments," ACM Computing Surveys (CSUR), vol. 31, no. 2, pp. 117-157, June 1999.